

Minimalist Grammars and Minimalist Categorial Grammars, definitions toward inclusion of generated languages

Maxime Amblard¹

Nancy Université - INRIA Nancy-Grand Est
amblard@loria.fr

Abstract. Stabler proposes an implementation of the Chomskyan Minimalist Program, [1] with Minimalist Grammars - MG, [2]. This framework inherits a long linguistic tradition. But the semantic calculus is more easily added if one uses the Curry-Howard isomorphism. Minimalist Categorial Grammars - MCG, based on an extension of the Lambek calculus, the mixed logic, were introduced to provide a theoretically-motivated syntax-semantics interface, [3]. In this article, we give full definitions of MG with algebraic tree descriptions and of MCG, and take the first steps towards giving a proof of inclusion of their generated languages.

The Minimalist Program - MP, introduced by Chomsky, [1], unified more than fifty years of linguistic research in a theoretical way. MP postulates that a *logical form* and a *sound* could be derived from *syntactic relations*. Stabler, [2], proposes a framework for this program in a computational perspective with Minimalist Grammars - MG. These grammars inherit a long tradition of generative linguistics. The most interesting contribution of these grammars is certainly that the derivation system is defined with only two rules: *merge* and *move*. The word *Minimalist* is introduced in this perspective of simplicity of the definitions of the framework. If the *merge* rule seems to be classic for this kind of treatment, the second rule, *move*, accounts for the main concepts of this theory and makes it possible to modify relations between elements in the derived structure.

Even if the phonological calculus is already defined, the logical one is more complex to express. Recently, solutions were explored that exploited Curry's distinction between tectogrammatical and phenogrammatical levels; for example, Lambda Grammars, [4], Abstract Categorial Grammars, [5], and Convergent Grammars [6]. First steps for a convergence between the Generative Theory and Categorial Grammars are due to S. Epstein, [7]. A full volume of *Language and Computation* proposes several articles in this perspective, [8], in particular [9], and Cornell's works on links between Lambek calculus and Transformational Grammars, [10]. Formulations of Minimalist Grammars in a Type-Theoretic way have also been proposed in [11], [12], [13]. These frameworks were evolved in [14], [3], [15] for the syntax-semantics interface.

Defining a syntax-semantics interface is complex. In his works, Stabler proposes to include this treatment directly in MG. But interactions between syntax

and semantic properties occur at different levels of representation. One solution is to suppose that these two levels should be synchronized. Then, the Curry-Howard isomorphism could be invoked to build a logical representation of utterances. The Minimalist Categorical Grammars have been defined from this perspective: capture the same properties as MG and propose a synchronized semantic calculus. We will propose definitions of these grammars in this article. But do MG and MCG generate the same language? In this article we take the first steps towards showing that they do.

The first section proposes new definitions of Minimalist Grammars based on an algebraic description of trees which allows to check properties of this framework, [3]. In the second section, we will focus on full definitions of Minimalist Categorical Grammars (especially the phonological calculus). We will give a short motivation for the syntax-semantics interface, but the complete presentation is delayed to a specific article with a complete example. These two parts should be viewed as a first step of the proof of mutual inclusion of languages between MG and MCG. This property is important because it enables us to reduce MG's to MCG, and we have a well-defined syntax-semantics interface for MCG.

1 Minimalist Grammars

Minimalist Grammars were introduced by Stabler [2] to encode the Minimalist Program of Chomsky, [1]. They capture linguistic relations between constituents and build trees close to classical Generative Analyses.

These grammars are fully lexicalized, that is to say they are specified by their lexicon. They are quite different from the traditional definition of lexicalized because they allow the use of specific items which do not carry any phonological form. The use of these items implies that MG represent more than syntactic relations and must be seen as a meta-calculus lead by the syntax.

These grammars build trees with two rules: *merge* and *move* which are triggered by features. This section presents all the definitions of MG in a formal way, using algebraic descriptions of trees.

1.1 Minimalist Tree Structures

To provide formal descriptions of Minimalist Grammars, we differ from traditional definitions by using an algebraic description of trees: a sub-tree is defined by its context, as in [16] and [17]. For example, the figure on the left of the figure 1 shows two subtrees in a tree (t_1 and t_2) and their context (C_1 and C_2). Before we explain the relations in minimalist trees, we give the formal material used to define a tree by its context.

Graded alphabets and trees: Trees are defined from a *graded set*. A graded set is made up of a support set, noted Σ , the alphabet of the tree, and a *rank* function, noted σ , which defines node arity (the *graded* terminology results from the rank function). In the following, we will use Σ to denote a graded (Σ, σ) .

The set of trees built on Σ , written T_Σ , is the smallest set of strings $(\Sigma \cup \{ (; ,) \})^*$. A leaf of a tree is a node of arity 0, denoted by α instead of $\alpha()$. For a tree t , if $t = \sigma(t_1, \dots, t_k)$, the root node of t is written σ .

Moreover, a set of variables $X = \{x_1, x_2, \dots\}$ is added for these trees. X_k is the set of k variables. These variables mark positions in trees. By using variables, we define a substitution rule: given a tree $t \in T_{\Sigma(X_k)}$ (i.e. a tree which contains instances of k variables x_1, \dots, x_k) and t_1, \dots, t_k , k trees in T_Σ , the tree obtained by simultaneous substitution of each instance of x_1 by t_1, \dots, x_k by t_k is denoted by $t[t_1, \dots, t_k]$. The set of all subtrees of t is noted \mathcal{S}_t .

Thus, for a given tree t and a given node n of t , the subtree for which n is the root is denoted by t with this subtree replaced by a variable.

Minimalist trees are produced by Minimalist Grammars and they are built on the graded alphabet $\{<, >, \Sigma\}$, whose ranks of $<$ and $>$ are 2 and 0 for strings of Σ . Minimalist Trees are binary ones whose nodes are labelled with $<$ or $>$, and whose leaves contain strings of Σ .

Relations between sub-trees We formalise relations for different positions of elements in \mathcal{S}_t . Intuitively, these define the concept of *be above*, *be on the right* or *on the left*. A specific relation on minimalist trees is also defined: *projection* that introduces the concept of *be the main element* in a tree.

In the following, we assume a given graded alphabet Σ . Proofs of principal properties and closure properties are all detailed in [3]. The first relation is the dominance which informally is the concept of *be above*.

Definition 1 Let $t \in T_\Sigma$, and $C_1, C_2 \in \mathcal{S}_t$, C_1 **dominates** C_2 (written $C_1 \triangleleft^* C_2$) if there exists $C' \in \mathcal{S}_t$ such that $C_1[C'] = C_2$.

Figure 1 shows an example of dominance in a tree. One interesting property of this algebraic description of trees is that properties in sub-trees pass to tree. For example, in a given tree t , if there exists C_1 and C_2 such that $C_1 \triangleleft^* C_2$, using a 1-context C , we could build a new tree $t' = C[t]$ (substitution in the position marked by the variable x_1 of t). Then, $C[C_1]$ and $C[C_2]$ exist (they are part of t') such that $C[C_1] \triangleleft C[C_2]$.

Definition 2 Let $t \in T_\Sigma$, $C_1, C_2 \in \mathcal{S}_t$, C_1 **immediately precedes** C_2 (written $C_1 \prec C_2$) if there exists $C \in \mathcal{S}_t$ such that:

1. $C_1 = C[\sigma(t_1, \dots, t_j, x_1, t_{j+2}, \dots, t_k)]$ and
2. $C_2 = C[\sigma(t_1, \dots, t_j, t_{j+1}, x_1, \dots, t_k)]$.

Precedence, written \prec^\sim , is the smallest relation defined by the following rules (transitivity rule, closure rule and relation between dominance and precedence relation):

$$\frac{C_1 \prec^\sim C_2 \quad C_2 \prec^\sim C_3}{C_1 \prec^\sim C_3} [trans] \quad \frac{C_1 \prec C_2}{C_1 \prec^\sim C_2} [*] \quad \frac{C_1 \triangleleft^* C_2}{C_2 \prec^\sim C_1} [dom]$$

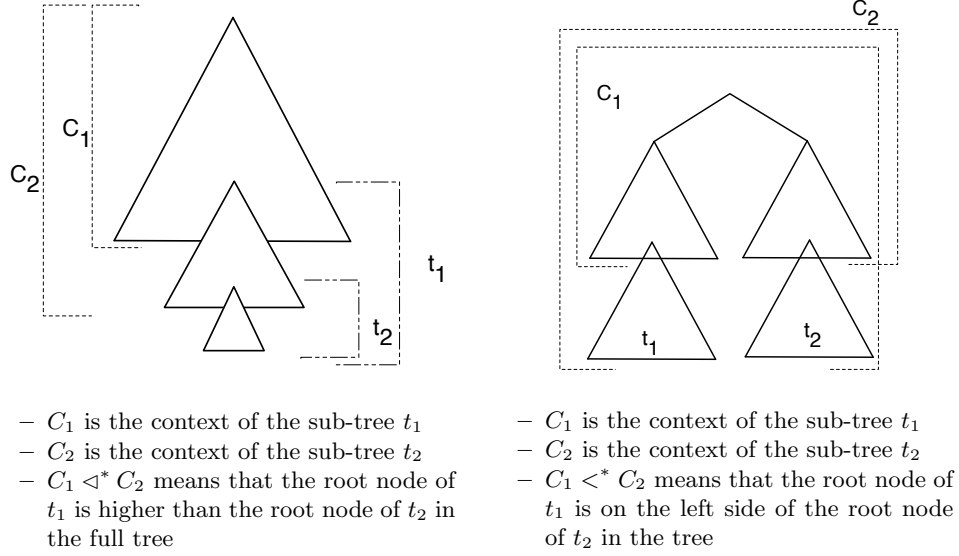


Fig. 1. Dominance and precedence relations in trees.

Precedence encodes the relation *be on the left* (and then *be on the right*) or *be above* another element (using the dominance). These two relations stay true for substitution (as mentioned above).

The next relation does not define a tree relation. It realises a linguistic property by leading the concept of *be the main* element in a structure (or a substructure).

Definition 3 Let $t \in T_{\Sigma_{MG}}(A)$, and $C_1, C_2 \in \mathcal{S}_t$, C_1 **immediately projects** on C_2 (written $C_1 < C_2$) if there exists $C \in \mathcal{S}_t$ such that one of the two following properties holds:

1. $C_1 = C[<(x_1, t_2)]$ and $C_2 = C[<(t_1, x_1)]$,
2. $C_1 = C[>(t_2, x_1)]$ and $C_2 = C[>(x_1, t_1)]$,

in this case $C \triangleleft C_1$ and $C \triangleleft C_2$. If $C_1 < C_2$ or $C_2 < C_1$, then there exists C such that $C \triangleleft C_1$ and $C \triangleleft C_2$.

$<\sim$ is the smallest relation defined by the following system of rules:

$$\begin{array}{c}
 \frac{C \in \mathcal{S}_t}{C <\sim C} [0] \quad \frac{C_1 <\sim C_2 \quad C_2 <\sim C_3}{C_1 <\sim C_3} [trans] \quad \frac{C_1 < C_2}{C_1 <\sim C_2} [\sim] \\
 \\
 \frac{C_1 \triangleleft^* C_2 \quad C_3 \triangleleft^* C_4 \quad C_2 < C_3}{C_1 <\sim C_4} [A] \quad \frac{C_1 \triangleleft C_2 \quad C_2 < C_3}{C_2 <\sim C_1} [B]
 \end{array}$$

Note that the projection relation is transitive. All the properties of these three relations are proven in [3]. The figure 2 presents three minimalist trees where in t the *main* element is the verb *walks* (which is accessible by following the projection relation).

These three relations could seem quite complicated for a reader who is not familiar with these notations or the zipper theory. But their expressiveness allows to prove the structural properties assumed for MG and moreover to give the proof of languages inclusion with MCG. Finally, in this section, we have defined the concept of parent and child relations in trees plus the projection relation which defines constituents in linguistic descriptions.

1.2 Linguistic Structures in Trees

From the linguistic perspective, trees represent relationships between grammatical elements of an utterance. Linguistic concepts are associated with minimalist tree structures. These relationships have been proposed for the analysis of structural analogies between verbal and nominal groups. Thus, groups of words in a coherent statement (phrases), whatever their nature, have a similar structure. This is supposed to be the same for all languages, regardless of the order of sub-terms. This assumption is one of the basic ideas of the X -bar theory introduced in the seventies, [18] and in the MP, [1].

The head is the element around which a group is composed. An easy way to find the head of a minimalist tree is to follow the projection relation of the nodes.

Definition 4 Let $t \in T_{MG}$, if for all $C' \in S_t$, $C' <^{\sim} C'$ then C is called the **head** of t . For a given tree $t \in T_{MG}$, we write $H_t[x] \in S_t$ a sub-tree of t of which x is the head, and $head(t)$ is a leaf which is the head of t . Then $t = H_t[head(t)]$.

For a minimalist tree, there always exists a unique minimal element for the projection relation and it is a leaf (which is the head of the tree) [3].

For example, the head of the minimalist tree in figure 2 is the leaf *walks* (follow the direction of the projection relation in nodes and stop in a leaf). Subtrees have their own head, for example the leaf *a* is the head of the subtree t_1 (in figure 2) and the preposition *in* is the head of t_3 .

Maximal Projection is, for a leaf l , the largest subtree for which l is the head. This is the inverse notion of *head*. In the minimalist tree of figure 2, the maximal projection of the leaf *walks* is the full tree t . To describe other maximal projections in this example, the maximal projection of *a* is the subtree which contains *a man* and the maximal projection of the *man* is the leaf *man*. In a more formal way, the maximal projection is defined as follows:

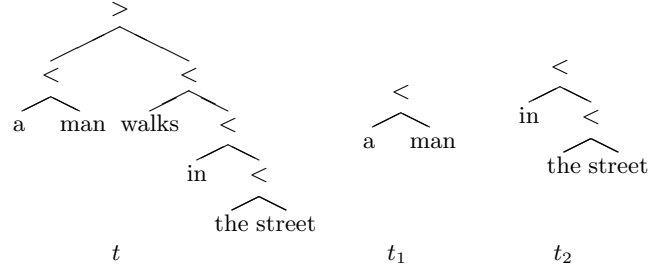


Fig. 2. a minimalist tree t and two of its sub-tree

Definition 5 Let $t \in T_{MG}$, $C \in S_t$. The **maximal projection** of C (denoted by $proj_{max}(C)$) is the subtree defined by:

- if $C = x_1$, $proj_{max}(C) = x_1$
- if $C = C'[\prec (x_1, t)]$ or $C = C'[\succ (t, x_1)]$, $proj_{max}(C) = proj_{max}(C')$
- if $C = C'[\prec (t, x_1)]$ or $C = C'[\succ (x_1, t)]$, $proj_{max}(C) = C$

Then $proj_{max}(walks) = t$. This logical characterization of minimalist trees and structural relations allows to prove different properties of MG (for example that the projection is anti-symmetric), [3].

Complement and Specifier are relations on subtrees with respect to the head.

Elements coming after the head provide information and they are in the *complement* relation. Let $t \in S_{MG}$, C_1 is a complement of $head(t) = C$, if $proj_{max}(C) \prec^* C_1$ and $C \prec^+ C_1$, denoted by $C_1 \text{ comp } C$.

In the tree t of figure 2, the subtree t_2 is in a complement relation with the head *walks*. It *adds* information to the verb.

By contrast, elements placed before the head determine who (or what) is in the relationship. Let $t \in S_{MG}$, C_1 is a specifier of $head(t) = C$, if $proj_{max}(C) \prec^* C_1$ and $C_1 \prec^+ C$, denoted by $C_1 \text{ spec } C$.

In the tree t of figure 2, the subtree t_1 is in a specifier relation with the head *walks*. It *specifies* interpretation of the verb.

1.3 Minimalist Grammars

The computational system of MG is entirely based on features which represent linguistic properties of constituents. Rules are triggered by these features and they build minimalist trees. A *Minimalist Grammar* is defined by a quintuplet $\langle V, Features, Lex, \Phi, c \rangle$ where:

- V is a finite set of non-syntactic features, which contains two sets: P (phonological forms, marked with / /), and I (logical forms, marked with ()).
- $Features = \{B \cup S \cup L_a \cup L_e\}$ is a finite set of syntactic features,
- Lex is a set of complex expressions from P and $Features$ (lexical items),
- $\Phi = \{merge, move\}$ is the set of generative rules,

- $c \in \text{Features}$ is the feature which allows to accept derivations.

The final tree of a derivation which ends with acceptance is called a *derivational tree*, which corresponds to a classical generative analysis. Phonological forms are used as lexical items (and they could be seen as the grammar's terminal symbols). A left-to-right reading of phonological forms in derived and accepted structures provides the recognized string. But intermediate trees in a derivation do not stand for this. Only the derivational tree allows to recognize a string. This results from the *move* rule which modifies the tree structure. For a MG G , the language L_G recognized by G is the closure of the lexicon by the generation rules.

1.4 Features

A MG is defined by its lexicon which stores its resources. Lexical items consist of a phonological form and a list of syntactic features. The syntactic set of features is divided in two subsets: one for basic categories, denoted B , and one for *move* features, denoted D . Different types of features are:

- $B = \{v, dp, c, \dots\}$ the set of **basic features**. Elements of B denote standard linguistic categories. Note that this set contains c , the *accepting feature* (I assume it is unique at least).
- $S = \{=d \mid d \in B\}$ the set of **selectors** which expresses the necessity of another feature of B of the same type (for $d \in B$, $=d$ is the dual selector).
- $L_a = \{+k \mid k \in D\}$ the set of **licensors**. These features assign an expression's property to complement another in a specifier-head relation.
- $L_e = \{-k \mid k \in D\}$ the set of **licensees**. These features state that the expression needs to be complemented by a similar licensor.

Lexical sequences of features follow the syntax: $/FP/ : (S(S \cup L_a)^*)^* B(L_e)^*$

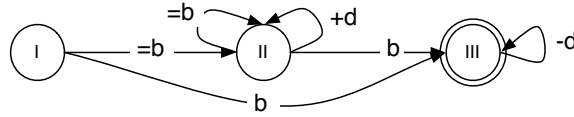


Fig. 3. Automata of acceptable sequences of features where $b \in B$ and $d \in D$.

Vermaat, [19], proposes an automata which recognises the acceptable sequences, proposed in figure 3. This structure could be divided in two parts: the first containing a sequence of selectors and licensors (features which trigger rules, as we shall see), and the second which contains only one basic feature (the grammatical category associated to the expression) and a sequence of licensees. The

first part corresponds to stat I and II and the second to stat III and transitions to this state. In the following, e will denote any feature and E a sequence of features (possibly empty).

For example, the sequence associated with an intransitive verb will be: $=d +case v$ which means that this verb must be jointed with a *determinal phrase* (*determinal* comes from the Generative Theory), a complex expression with feature d . Then it must be combined with a $-case$, we will see how in the next section, and then there is a structure associated with *verb* (feature v).

Transitive verbs will extend the intransitive ones with the list:

$$=d +case =d +case v$$

The two $=d$ correspond to the subject and the object of the verb. The first *case* will be *accusative* and the second *nominative*.

Another example is determiners: they are combined with a noun to build a determiner phrase and need to be unified in the structure (see the next section). Here is an example of lexicon which contains a verb, a noun and a determiner:

$$\begin{aligned} walks &: =d +case v \\ a &: =n d -case \\ man &: n \end{aligned}$$

1.5 MG Rules

Φ , the set of generating rules, contains only: *merge* and *move*. A derivation is a succession of rule applications which build trees. These trees are partial results: the structural order of phonological forms does not need to correspond to the final one. In the MP, a specific point, called *Spell-Out* is the border between the calculus of derivations and the final result. Rules are triggered by the feature occurring as the first element of list of features of the head.

Merge is the process which connects different parts. It is an operation which joins two trees to build a new one:

$$merge : T_{MG} \times T_{MG} \rightarrow T_{MG}$$

It is triggered by a selector ($=x$) at the top of the list of features of the head and it is realised with a corresponding basic feature (x) at the top of the list of features of the head of a second tree. *Merge* adds a new root which dominates both trees and cancels the two features. The specifier/complement relation is implied by the lexical status of the tree which carried the selector. The new root node points to this tree.

Let $t, t' \in T_{MG}$ be such that $t = H_t[l : =h E]$ and $t' = H_{t'}[l' : h E']$ with $h \in B$:

$$merge(t, t') = \begin{cases} < (l : E, H_{t'}[l' : E']) & \text{if } t \in Lex, \\ > (H_{t'}[l' : E'], H_t[l : E]) & \text{otherwise.} \end{cases}$$

Figure 4 presents the graphical representation of *merge*.

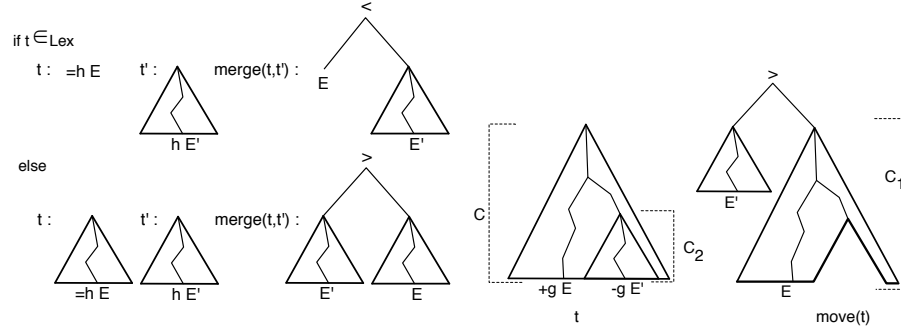
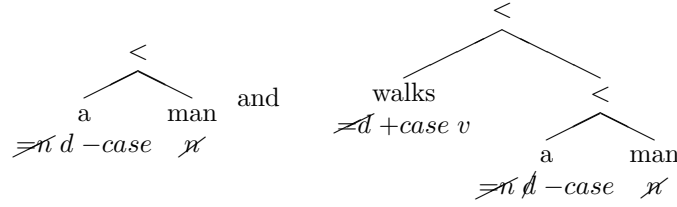


Fig. 4. Tree representation of *merge* and *move*.

For example, to derive *a man walks*, we first need to combine *a* with *man*, and then to combine the result with the verb:



Obtained trees do not verify the word order (only the final tree will check the right word order). In this example, the selectors are carried by lexical items, then projection relations point to the left in both cases.

Move encodes the main idea of the Minimalist Program. It corresponds to the movement of a constituent at the top position in a derivation. *Move* is triggered by a licenser ($+x$) at the top of the list of features of the head of a tree. Then, it looks for a corresponding licensee ($-x$) at the top of the list of features of the head inside the tree. If these conditions are met, the maximal projection of the node which carries the licensee is moved to the left of a new root. This node points to the right (the subtree which carries the former head). Both licenser and licensee are cancelled. The root of the moved maximal projection is substituted by an empty leaf (ϵ). This new leaf is called the *trace* of the move.

Figure 4 shows a graphical representation of the move rule where the head of C carries a $+g$ in its top features list. Then we look for a leaf with $-g$ in its top features list and then find its maximal projection (C_2) which contains all the elements which depend on it. Finally this sub-tree is moved to the left position of a new root node. Intuitively, a linguistic property is checked and the consequence is a move in first position in the tree. And strictly:

$$move : T_{MG} \rightarrow T_{MG}$$

For all tree $t = C[l : +g E, l' : -g E']$, such that $t = H_t[l : +g E]$, there exists $C_1, C_2 \in S_t$ such that: C_2 is the maximal projection of the leaf l' and C_1 is t deprived of C_2 . Then, $t = C_1[l : +g E, C_2[l' : -g E']]$ where:

- $C_2[l' : -g E'] = proj_{max}(C[l' : -g E])$
- $C_1[l : +g E, x_1] = proj_{max}(C[l : +g E, x_1])$

$$move(t) = >(C_2[l' : E'], C_1[l : E, \epsilon])$$

Figure 4 presents the graphical representation of *move*.

Stabler introduces some refinements to these grammars. Let us mention them. He introduces a second *move*: *weak move*, which does not move the phonological forms. The precedent *move* is then called *strong move*, which is triggered with capital features. The *weak move* is, like *strong move*:

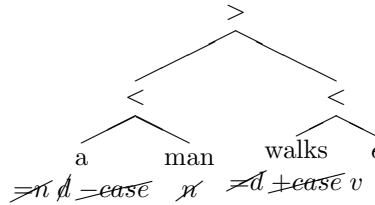
$$move(t) = >(C_2[\epsilon : E'], C_1[l : E, l'])$$

Variations on *strong/weak* values achieve variations on phonological order. This is an instance of the use of *parameters* of the Minimalist Program.

Moreover, restrictions can be introduced on MG derivations. An important one is the Shortest Move Condition (SMC) which blocks *move* in case of ambiguity on licensees. Then, the move operation of MG with SMC is deterministic.

A locality condition could also be introduced: Specifier Island Condition - SPIC. “Islands” define areas which prohibit extractions. With SPIC, a subtree cannot be moved if it is in a specifier relation within a subtree. This condition was introduced by Stabler, in [20] drawing on works of [21] and [22], who proposes that moved elements had to be in a complement relation.

In the previous example, the head of the last tree is the leaf *walks* which contains a *+case* feature as first element of its list. Then, a *move* is triggered in the tree with the leaf *a* which carries a (*-case*). The resulting tree is the following:



The move operation modifies the position of the maximal projection of the leaf which carries the *-case*. The old position is substituted by an empty leaf (ϵ). Finally, the tree contains only one feature which is *v*. In this small example, I did not discuss the validity of the final feature, but in a real derivation, we assume that it is not the verb which carries the *+case* licenser which corresponds to the nominal case, but it is a specific item. This item corresponds to the morphological mark of the verb. Then each acceptable derivation assumes that a verb has received its time (and other properties). But exhibiting the use of

this item needs other refinements of the two rules (Head-movement and Affix-Hopping).

This section did not propose a new framework for computational linguistics. This is a new definition of Stabler proposal. This way, assumed properties of minimalist trees have been fully proved, [3]. Moreover this algebraic definition of MG is a perfect description to compare generated languages with other frameworks. Finally, this modifies the point of view on derivations and shows all steps of the calculus as substitution. One missing point is still the introduction of a semantic calculus. Let us now develop MCG which are defined with a syntax-semantics interface.

2 Minimalist Categorical Grammars - MCG

In this section, we define a new Type-Theoretic Framework which is provided by the mixed calculus, a formulation of Partially Commutative Linear Logic. It proposes to simulate MG and then keep linguistic properties of the Minimalist Program. MCG are motivated by the syntax-semantics interface, [3]. This interface, as for Lambek calculus, is based on an extension of the Curry-Howard isomorphism, [23]. Even though this interface is not the aim of this paper, let us discuss some important points.

The idea of encoding MP with Lambek calculus arises from [11] and extended versions of this work. In these propositions, the calculus is always non-commutative, a property needed to model the left-right relation in sentences. But the *move* operation could not be defined in a proper way with non-commutative relation. In particular, in complex utterances, the non-commutativity implies that a constituent (for example the object DP) must be fully treated before another one is introduced (for example the subject DP). Otherwise, features are mixed and non-commutativity blocks resolutions. It is not acceptable to normalize the framework with such a strong property and it makes the system inconsistent in regard to linguistics.

The solution we propose is to define a new framework which allows to deal with commutative and non-commutative connectors: the mixed calculus. The main consequence on the model of this calculus is that variables in logical formulae are introduced at different places and must be unified later. In [3] we show how the unification is used to capture semantic phenomena which are not easily included. In few words, the idea is to consider proofs of mixed calculus as *phases* of a verb. Phases have been introduced by Chomsky to detail different modifications which occur on a verb. Several linguists have showed that phases have implications on semantics, for example the theta-roles must be allocated after a specific phase. This is exactly the result of the syntax-semantics interface of MCG. Full explanations need more space to be presented, but the main contribution of MCG is to propose an efficient syntax-semantics interface in the same perspective as MG.

In this section, we will detail MCG and expose their structural link with MG. First we present the mixed calculus, then we give definitions of MCG and show

proofs of the mixed calculus produced by MCG (together with their linguistic properties).

2.1 Mixed calculus

MCG are provided with mixed calculus, [24], a formulation of Partially Commutative Linear Logic. Hypotheses are either in a non-commutative order ($<$; $>$) or in a commutative one $((,))$. The plain calculus contains introduction and elimination rules for:

- the non-commutative product \odot :

$$\frac{\Delta \vdash A \odot B \quad \Gamma, < A; B >, \Gamma' \vdash C}{\Gamma, \Delta, \Gamma' \vdash C} [\odot_e] \quad \frac{\Delta \vdash A \quad \Gamma \vdash B}{< \Delta; \Gamma > \vdash A \odot B} [\odot_i]$$

- its residuals ($/$ and \backslash):

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \backslash C}{< \Gamma; \Delta > \vdash C} [\backslash_e] \quad \frac{\Delta \vdash A / C \quad \Gamma \vdash A}{< \Delta; \Gamma > \vdash C} [/_e]$$

$$\frac{< A; \Gamma > \vdash C}{\Gamma \vdash A \backslash C} [\backslash_i] \quad \frac{< \Gamma; A > \vdash C}{\Gamma \vdash C / A} [/_i]$$

- the commutative product \otimes :

$$\frac{\Delta \vdash A \otimes B \quad \Gamma, (A, B), \Gamma' \vdash C}{\Gamma, \Delta, \Gamma' \vdash C} [\otimes_e] \quad \frac{\Delta \vdash A \quad \Gamma \vdash B}{(\Delta, \Gamma) \vdash A \otimes B} [\otimes_i]$$

- its residual \multimap :

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \multimap C}{(\Gamma, \Delta) \vdash C} [\multimap_e] \quad \frac{(A, \Gamma) \vdash C}{\Gamma \vdash A \multimap C} [\multimap_i]$$

The product connectors of the mixed calculus use in a first step hypotheses to mark positions in the proof and in a second one substitute the result of an another proof in these positions using a product elimination (the commutative/non-commutative status depends on relations between hypotheses). This is exactly the process we will use to define the *move* rule of MCGs.

Moreover, the calculus contains an axiom rule and an entropy rule. This last one allows to relax the order between hypotheses. We will use this rule to define *merge* in MCG as we will see in the following section.

$$\frac{}{A \vdash A} [axiom] \quad \frac{\Gamma \vdash C}{\Gamma' \vdash C} [\text{entropy} \text{ — whenever } \Gamma' \sqsubset \Gamma]$$

This calculus has been shown to be normalizable, [25] and derivations of MCG will be proofs of the mixed calculus in normal form.

2.2 Minimalist Categorical Grammars

As MG, MCG are lexicalized grammars. Derivations are led by formulae associated with lexical items built with connectors of the mixed logic. They are specific proofs of the mixed logic, labelled to realise the phonological and semantic tiers. Phonological labels on proofs will be presented with definitions of MCG rules.

A MCG is defined by a quintuplet $\langle N, P, Lex, \Phi, C \rangle$ where :

- N is the union of two finite disjoint sets Ph and I which are respectively the *set of phonological forms* and the one of *logical forms*.
- P is the union of two finite disjoint sets P_1 and P_2 which are respectively the *set of constituent features* (the set B of MG) and the one of *move features* (the set D of MG).
- Lex is a finite subset of $E \times F \times I$, the set of lexical items ¹.
- $\Phi = \{merge, move\}$ is the set of generative rules,
- $C \in P$ is the accepting formulae.

As mentioned in the previous section, *move* is defined using a product elimination. In MG, a constituent is first introduced in a tree using its basic feature and then can be moved using its licensees. In MCG, a constituent will be introduced only when all its positions (which correspond to the basic feature and its licensees) have been marked in the proof by specific hypotheses. But we need to distinguish the type of the basic feature from the licensees features. That is why P is divided in two subsets P_1 and P_2 . This sub-typing of formulae is used to well define lexicons of MCG.

The set E is Ph^* , and the set F , the set of formulae used to build Lex , is defined with the set P , the commutative product \otimes and the two non-commutative implications $/$ and \backslash . Formulae of F are recognized by the non-terminal L of the following grammar:

$$\begin{aligned} L &::= (B) / P_1 \mid C \\ B &::= P_1 \backslash (B) \mid P_2 \backslash (B) \mid C \\ C &::= P_2 \otimes (C) \mid C_1 \\ C_1 &::= P_1 \end{aligned}$$

In more details, MCG formulae start with a $/$ which is followed by a sequence of \backslash . This sequence contains operators allowing to compose the proof with another one (operators are the translation of selectors and licensors). Lexical formulae are ended by a sequence of \otimes . To sum up, these formulae have the structure $(c_m \backslash \dots \backslash c_1 \backslash (b_1 \otimes \dots \otimes b_n \otimes a)) / d$, with $a \in P_1$, $b_i \in P_2$, $c_j \in P$ and $d \in P_1$. This structure corresponds to the two parts of the list of features we have mentioned in the previous section.

For the example *a man walks*, the MCG's lexicon is the following:

$$\begin{aligned} \text{walks} &: case \backslash v / d \\ a &: (case \otimes d) / n \\ \text{man} &: n \end{aligned}$$

Licensees, which express the need for an information, are there seen as a specific part of the basic feature (a part of the main sub-type). Licensors will be cancelled with an hypothesis to mark a position in the proof. Distinction between them is not written by an *ad hoc* marker but by structural relations inside the formula. Before we explain the *move* and *merge* rules, let us present the phonological tiers.

¹ In the following, Lex is a subset of $E \times F$. The semantic part is used for the syntax-semantics interface which is not detailed here.

2.3 Derivations

Labels. Derivations of MCG are labelled proofs of the mixed calculus. Before defining labelling, we define labels and operations on them.

Let V be an uncountable and finite set of variables such that: $Ph \cap V = \emptyset$. T is the union of Ph and V . We define the set Σ , called *labels set* as the set of triplets of elements of T^* . Every position in a triplet has a linguistic interpretation: they correspond to specifier/head/complement relations of minimalist trees. A label r will be considered as $r = (r_{spec}, r_{head}, r_{comp})$.

For a label in which there is an empty position, we adopt the following notation: $r_{-head} = (r_{spec}, \epsilon, r_{comp})$, $r_{-spec} = (\epsilon, r_{head}, r_{comp})$, $r_{-comp} = (r_{spec}, r_{head}, \epsilon)$. We introduce variables in the string triplets and a substitution operation. They are used to modify a position inside a triplet by a specific material. Intuitively, this is the counterpart in the phonological calculus of the product elimination. The set of variables with at least one in r is denoted by $Var(r)$. The number of occurrences of a variable x in a string $s \in T^*$ is denoted by $|s|_x$, and the number of occurrences of x in r by $\varphi_x(r)$. A label is *linear* if for all x in V , $\varphi_x(r) \leq 1$.

A *substitution* is a partial function from V to T^* . For σ a substitution, s a string of T^* and r a label, we note $s.\sigma$ and $r.\sigma$ the string and the label obtained by the simultaneous substitution in s and r of the variables by the values associated by σ (variables for which σ is not defined remain the same).

If the domain of definition of a substitution σ is finite and equal to x_1, \dots, x_n and $\sigma(x_i) = t_i$, then σ is denoted by $[t_1/x_1, \dots, t_n/x_n]$. Moreover, for a sequence s and a label r , $s.\sigma$ and $r.\sigma$ are respectively denoted $s[t_1/x_1, \dots, t_n/x_n]$ and $r[t_1/x_1, \dots, t_n/x_n]$. Every injective substitution which takes values in V is called *renaming*. Two labels r_1 and r_2 (respectively two strings s_1 and s_2) are equal modulo a renaming of variables if there exists a renaming σ such that $r_1.\sigma = r_2$ (*resp.* $s_1.\sigma = s_2$).

Finally, we need another operation on string triplets which allows to combine them together: the string concatenation of T^* is noted \bullet . Let *Concat* be the operation of concatenation on labels which concatenates the three components in the linear order: for $r \in \Sigma$, $Concat(r) = r_{spec} \bullet r_{head} \bullet r_{comp}$.

We then have defined a phonological structure which encodes specifier/complement/head relations and two operations (substitution and concatenation). These two operations will be counterparts in the phonological calculus of *merge* and *move*.

Labelled proofs. Before exhibiting the rules of MCG, the concept of labelling on a subset of rules of the mixed logic is introduced. *Minimalist logic* is the fragment of mixed logic composed by the axiom rule, \backslash_e , $/_e$, \otimes_e and \sqsubset .

For a given MCG $G = \langle N, P, Lex, \Phi, C \rangle$, let a *G-background* be $x : A$ with $x \in V$ and $A \in F$, or $\langle G_1; G_2 \rangle$ or else (G_1, G_2) with G_1 and G_2 some *G-backgrounds* which are defined on two disjoint sets of variables. *G-backgrounds* are series-parallel orders on subsets of $V \times F$. They are naturally extended to the entropy rule, noted \sqsubset . A *G-sequent* is a sequent of the form: $\Gamma \vdash_G (r_s, r_t, r_c) : B$ where Γ is a *G-background*, $B \in F$ and $(r_s, r_t, r_c) \in \Sigma$.

A *G*-labelling is a derivation of a *G*-sequent obtained with the following rules:

$$\begin{array}{c}
\frac{\langle s, A \rangle \in Lex}{\vdash_G (\epsilon, s, \epsilon) : A} [Lex] \\
\\
\frac{x \in V}{x : A \vdash_G (\epsilon, x, \epsilon) : A} [axiom] \\
\\
\frac{\Gamma \vdash_G r_1 : A / B \quad \Delta \vdash_G r_2 : B \quad Var(r_1) \cap Var(r_2) = \emptyset}{\langle \Gamma; \Delta \rangle \vdash_G (r_{1s}, r_{1t}, r_{1c} \bullet Concat(r_2)) : A} [/_e] \\
\\
\frac{\Delta \vdash_G r_2 : B \quad \Gamma \vdash_G r_1 : B \setminus A \quad Var(r_1) \cap Var(r_2) = \emptyset}{\langle \Gamma; \Delta \rangle \vdash_G (Concat(r_2) \bullet r_{1s}, r_{1t}, r_{1c}) : A} [\setminus_e] \\
\\
\frac{\Gamma \vdash_G r_1 : A \otimes B \quad \Delta[x : A, y : B] \vdash_G r_2 : C \quad Var(r_1) \cap Var(r_2) = \emptyset \quad A \in P_2}{\Delta[\Gamma] \vdash_G r_2[Concat(r_1)/x, \epsilon/y] : C} [\otimes_e] \\
\\
\frac{\Gamma \vdash_G r : A \quad \Gamma' \sqsubset \Gamma}{\Gamma' \vdash_G r : A} [\sqsubset]
\end{array}$$

Note that a *G*-labelling is a proof tree of the minimalist logic on which sequent hypotheses are decorated with variables and sequent conclusions are decorated with labels. Product elimination is used with a substitution on labels and implication connectors with concatenation (a triplet is introduced in another one by concatenating its three components).

If $\Gamma \vdash_G r : B$ is a *G*-sequent derivable, then r is linear, and $Var(r)$ is exactly the set of variables in Γ . Finally, for all renamings σ , $\Gamma.\sigma \vdash_G r.\sigma : B$ is a *G*-sequent differentiable.

Merge and Move rules are simulated by combinations of rules of the minimalist logic producing *G*-labeling.

Merge is the elimination of $/$ (*resp.* \setminus) immediately followed by an *entropy* rule. The meaning of this rule is joining two elements in regard to the left-right order (then non-commutative connectors are used) and, as mentioned earlier, all hypotheses must be accessible. To respect this, a commutative order between hypotheses is needed. Then an entropy rule immediately follows each implication elimination.

For the phonological tier, a label is concatenated in the complement (respectively specifier) position in another one. Note that a *merge* which uses $/$ must be realized with a lexical item, so the context is always empty.

$$\begin{array}{c}
\frac{\vdash (r_{spec}, r_{head}, r_{comp}) : A / B \quad \Delta \vdash s : B}{\Delta \vdash (r_{spec}, r_{head}, r_{comp} \bullet Concat(s)) : A} [/_e] \\
\\
\frac{\Delta \vdash (r_{spec}, r_{head}, r_{comp} \bullet Concat(s)) : A}{\Delta \vdash (r_{spec}, r_{head}, r_{comp} \bullet Concat(s)) : A} [\sqsubset]
\end{array}$$

$$\frac{\frac{\Delta \vdash s : B \quad \Gamma \vdash (r_{spec}, r_{head}, r_{comp}) : B \setminus A}{\langle \Delta; \Gamma \rangle \vdash (Concat(s) \bullet r_{spec}, r_{head}, r_{comp}) : A} [\setminus_e]}{\Delta, \Gamma \vdash (Concat(s) \bullet r_{spec}, r_{head}, r_{comp}) : A} [\sqsubset]$$

These combinations of rules are noted $[mg]$.

For example, the proof of the utterance *a man walks* begins with the formulae of *walks: case \setminus v/d*. The first step of the calculus is to introduce two hypotheses, one for d and the other for *case*. The result is the following proof:

$$\frac{v : case \vdash (\epsilon, v, \epsilon) : case \quad \frac{\vdash (\epsilon, walks, \epsilon) : case \setminus v/d \quad u : d \vdash (\epsilon, u, \epsilon) : d}{u : d \vdash (\epsilon, walks, u) : case \setminus v} [mg]}{(v : case, u : d) \vdash (\epsilon, walks, u) : v} [mg]$$

In parallel, the derivation joins the determiner *a* and the noun *man*:

$$\frac{\vdash (\epsilon, a, \epsilon) : (case \otimes d)/n \quad \vdash (\epsilon, man, \epsilon) : n}{\vdash (\epsilon, a, man) : case \otimes d} [mg]$$

Note that the first proof contains two hypotheses which correspond to the type of the main formula in the second proof. The link between these two proofs will be made by a *move*, as we will show later.

Move is simulated by an elimination of a commutative product in a proof and, for the phonological calculus, is a substitution. We have structured the lexicons and the merge rule to delay to the move rule only the substitution part of the calculus.

$$\frac{\Gamma \vdash r_1 : A \otimes B \quad \Delta[u : A, v : B] \vdash r_2 : C}{\Delta[\Gamma] \vdash r_2[Concat(r_1)/u, \epsilon/v] : C} [\otimes_e]$$

This rule is applied only if $A \in P_2$ and B is of the form $B_1 \times \dots B_n \times D$ where $B_i \in P_2$ and $D \in P_1$.

This rule is noted $[mv]$. *Move* uses hypotheses as resources. The calculus places hypotheses in the proof, and when all hypotheses corresponding to a constituent are introduced, this constituent is substituted. The hypothesis P_1 is the first place of a moved constituent and hypotheses of P_2 mark the different places where the constituent is moved or have a trace.

In recent propositions, Chomsky proposes to delay all moves after the realisation of all merges. MCG could not encode this but contrary to MG where a *move* blocks all the process, in MCG *merge* could happen, except in the case of hypotheses of a given constituent shared by two proofs which must be linked by a *move*.

In our example, we have two proofs:

- one for the verb: $(v : case, u : d) \vdash (\epsilon, walks, u) : v$
- one for the DP: $\vdash (\epsilon, a, man) : case \otimes d$

The first hypothesis corresponds to the entry position of the DP in MG and the second to the moved position. Here, we directly introduce the DP by eliminating the two hypotheses in the same step:

$$\frac{\vdash (\epsilon, a, man) : case \otimes d \quad (v : case, u : d) \vdash (\epsilon, walks, u) : v}{\vdash (a \text{ man}, walks, \epsilon) : v} [mv]$$

The phonological result is *a man walks*. The proof encodes the same structure as the derivational tree of MG (modulo a small transduction on the proof).

For *cyclic move* (where a constituent is moved several times) all hypotheses inside this move must be linked together upon their introduction in the proof. For this, when a new hypothesis A is introduced, a $[mv]$ is applied with a sequent with hypothesis $A \otimes B \vdash A \otimes B$ where A is in P_2 and B is of the form $B_1 \otimes \dots \otimes B_n \otimes D$ where $B_i \in P_2$ and $D \in P_1$.

$$\frac{x : A \otimes B \vdash (\epsilon, x, \epsilon) : A \otimes B \quad \Delta[u : A, v : B] \vdash r : C}{\Delta[A \otimes B] \vdash r[x/u, \epsilon/v] : C} [\otimes_e]$$

In the definition of *merge*, the systematic use of entropy comes from the definition of *move*. As it was presented, *move* consumes hypotheses of the proof. But, from a linguistic perspective, these hypotheses could not be supposed introduced next to each other. The non-commutative order inferred from \backslash_e and $/_e$ blocks the *move* application. To avoid this, the entropy rule places them in commutative order. In MCG, all hypotheses are in the same relation, then to simplify the reading of proofs, the order is denoted only with $' , '$.

The strong/weak move could be simulated with the localization of the substitution (if hypotheses are in P_1 or P_2).

$$\frac{s : \Gamma \vdash A \otimes B \quad r[u, v] : \Delta[u : A, v : B] \vdash C}{r[Concat(s)/u, \epsilon/v] : \Delta[\Gamma] \vdash C} [move_{strong}]$$

$$\frac{s : \Gamma \vdash A \otimes B \quad r[u, v] : \Delta[u : A, v : B] \vdash C}{r[\epsilon/u, Concat(s)/v] : \Delta[\Gamma] \vdash C} [move_{weak}]$$

This version of move is quite different from the one presented for MG, but is close to one developed for later MG such as [26].

The main difference between MG and MCG comes from *move*: in MCG, constituents do not move but use hypotheses marking their places. MCG uses commutativity properties of mixed logic and see hypotheses as resources. To sum up, the derivation rules of MCG is the following set of rules:

$$\begin{array}{c}
\frac{\langle s, A \rangle \in Lex}{\vdash_G (\epsilon, s, \epsilon) : A} [Lex] \quad \frac{\vdash (r_{spec}, r_{head}, r_{comp}) : A / B \quad \Delta \vdash s : B}{\Delta \vdash (r_{spec}, r_{head}, r_{comp} \bullet Concat(s)) : A} [mg] \\
\\
\frac{\Delta \vdash s : B \quad \Gamma \vdash (r_{spec}, r_{head}, r_{comp}) : B \setminus A}{\Delta, \Gamma \vdash (Concat(s) \bullet r_{spec}, r_{head}, r_{comp}) : A} [mg] \\
\\
\frac{\Gamma \vdash r_1 : A \otimes B \quad \Delta[u : A, v : B] \vdash r_2 : C}{\Delta[\Gamma] \vdash r_2[Concat(r_1)/u, \epsilon/v] : C} [mv]
\end{array}$$

The set \mathbb{D}_G of recognized derivations by a MCG G is the set of proofs obtained with this set of rules and for which the concluding sequent is $\vdash r : C$. The language generated by G is $L(G) = \{Concat(r) \mid \vdash r : C \in \mathbb{D}_G\}$.

These derivations do not formally conserve the projection relation (nor the specifier, head and complement relations). These principles are reintroduced with strings. However, the head of a proof could be seen as the *principal formula* of mixed logic, and then by extension, the maximal projection is the proof for which a formula is the principal one. Specifier and complement are only elements on the right or left of this formula.

An interesting remark is that rules of MCG do not use the introduction rule of the mixed calculus. This way, they only try to combine together formulae extracted from a lexicon and hypotheses. As in MG where a derivation cancels features, the MCG system only consumes hypotheses and always reduces the size of the main formula (only the size of the context could increase). This corresponds to the cognitive fact that we stress the system in the analysis perspective. Introduction rules could be seen as captured by the given lexicon. But, because of the strong structure of the items, we directly associate formulae and strings.

We have presented all the MCG rules and lexicon, and illustrated them with a tiny example which encodes the main properties of this framework.

3 Conclusion

In this article, we propose new definitions of MG based on an algebraic description of trees. These definitions allow to check properties of this framework and moreover give a formal account to analyse links with other frameworks. Then, we give the definitions of MCG, a Type-Theoretic framework for MG. In this framework, *merge* and *move* are simulated by rules of the mixed logic (an extension of Lambek calculus to product and non-commutative connectors). The phonological calculus is added by labelling proofs of this logic.

The main contribution of MCG is certainly its syntax-semantics interface. This calculus is synchronized on proofs of MCG. But more technical details are needed to present this interface and the linguistic properties which it encodes. We delay the presentation of this interface to a future presentation.

Finally, the syntax-semantics interface of MCG should be used under the condition they keep properties of MG. This is the aim of another future article

which will present the proof of inclusion of MG generated languages in MCG generated languages. To prove this property, two alternative representations of MG and MCG derivations are introduced: *alternative derived structures* and *split proofs* and the corresponding *merge* and *move*. These structures and rules make the gap between the two kinds of derivations. They need technical details and more space to be presented.

Definitions and proofs could be easily extended to refinements of *merge: Affix-Hopping* and *Head-Movement* because these operations derived the same strings in both structures. But we have not included these rules in this presentation. On another hand, the proof of inclusion presented here does not include the SMC. The interpretation of SMC in MCG must be better defined before being included in such perspective. The generative power of these grammars with shortest move condition is still open.

This article is a first step to several perspectives which make a strong link between a well defined framework with many linguistic properties and a new one which captures this framework and proposes a syntax-semantics interface.

Acknowledgements

The author would like to express his deep gratitude to his supervisors Alain Lecomte and Christian Retoré. In particular, discussions with Alain Lecomte was a source of supports and good advices which turn the author to this research.

The author also wants to thank the associated editors and the anonymous reviewers for their constructive remarks and suggestions, and finally Patrick Blackburn and Mathieu Morey for their careful readings.

References

1. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
2. Stabler, E.: Derivational minimalism. *Logical Aspect of Computational Linguistic* **1328** (1997)
3. Amblard, M.: Calcul de représentations sémantiques et syntaxe générative: les grammaires minimalistes catégorielles. PhD thesis, université de Bordeaux 1 (septembre 2007)
4. Muskens, R.: Language, Lambdas, and Logic. In Kruijff, G.J., Oehrle, R., eds.: *Resource Sensitivity in Binding and Anaphora*. Studies in Linguistics and Philosophy. Kluwer (2003) 23–54
5. de Groote, P.: Towards abstract categorial grammars. Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference (2001)
6. Mansfield, L., Martin, S., Pollard, C., Worth, C.: Phenogrammatical labelling in convergent grammar: the case of wrap. unpublished (2009)
7. Berwick, R., Epstein, S.: On the convergence of 'minimalist' syntax and categorial grammars (1996)
8. Retoré, C., Stabler, E.: *Research on Language and Computation*. Volume 2(1). Christian Retoré and Edward Stabler (2004)

9. Lecomte, A.: Rebuilding the minimalist program on a logical ground. *Journal of Research on Language and Computation* **2(1)** (2004) 27–55
10. Cornell, T.: Lambek calculus for transformational grammars. *Journal of Research on Language and Computation* **2(1)** (2004) 105–126
11. Lecomte, A., Retoré, C.: Towards a logic for minimalist. *Formal Grammar* (1999)
12. Lecomte, A., Retoré, C.: Extending Lambek grammars: a logical account of minimalist grammars. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL 2001, Toulouse, ACL (July 2001)* 354–361
13. Lecomte, A.: Categorical grammar for minimalism. *Language and Grammar : Studies in Mathematical Linguistics and Natural Language* **CSLI Lecture Notes**(168) (2005) 163–188
14. Amblard, M., Lecomte, A., Retoré, C.: Syntax and semantics interacting in a minimalist theory. *Prospect and advance in the Syntax/Semantic interface* (October 2003) 17–22
15. Amblard, M., Lecomte, A., Retoré, C.: Synchronization syntax semantic for a minimalist theory. *Journée Sémantique et Modélisation* (mars 2004)
16. Huet, G.P.: The zipper. *J. Funct. Program* **7(5)** (1997) 549–554
17. Levy, J.J., Cori, R.: *Algorithmes et Programmation*. Ecole Polytechnique
18. Chomsky, N.: Conditions on transformations. In Kiparsky, S.A..P., ed.: *A Festschrift for Morris Halle*. Holt Rinehart and Winston (1973) 232–286
19. Vermaat, W.: *Controlling movement: Minimalism in a deductive perspective*. Master’s thesis, Universiteit Utrecht (1999)
20. Stabler, E.: Remnant movement and structural complexity. *Constraints and Resources in Natural Language Syntax and Semantics* (1999) 299–326
21. Koopman, H., Szabolcsi, A.: *A verbal Complex*. MIT Press, Cambridge (2000)
22. Kayne, R.S.: Overt vs covert movment. *Syntax* 1,2 (1998) 128–191
23. Howard, W.A.: The formulae-as-types notion of construction. In Hindley, J., Seldin, J., eds.: *To H.B. Curry: Essays on Combinatory Logic, λ -calculus and Formalism*. Academic Press (1980) 479–490
24. de Groote, P.: Partially commutative linear logic: sequent calculus and phase semantics. In Abrusci, V.M., Casadio, C., eds.: *Third Roma Workshop: Proofs and Linguistics Categories – Applications of Logic to the analysis and implementation of Natural Language*, Bologna:CLUEB (1996) 199–208
25. Amblard, M., Retore, C.: Natural deduction and normalisation for partially commutative linear logic and lambek calculus with product. *Computation and Logic in the Real World, CiE 2007 Quaderni del Dipartimento di Scienze Matematiche e Informatiche "Roberto Magari"* (june 2007)
26. Kobele, G.: *Generating Copies: An Investigation into Structural Identity in Language and Grammar*. PhD thesis, University of California, Los Angeles (2006)